



**HAL**  
open science

## Lower bounds for prams over $Z$

Luc Pellissier, Thomas Seiller

► **To cite this version:**

| Luc Pellissier, Thomas Seiller. Lower bounds for prams over  $Z$ . 2020. hal-02487667v1

**HAL Id: hal-02487667**

**<https://hal.science/hal-02487667v1>**

Preprint submitted on 21 Feb 2020 (v1), last revised 26 Jan 2021 (v2)

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Lower bounds for PRAMs over $\mathbb{Z}$

Luc Pellissier\*

LIX, Inria & École Polytechnique, France  
<https://lipn.univ-paris13.fr/~pellissier/>  
[luc.pellissier@inria.fr](mailto:luc.pellissier@inria.fr)

Thomas Seiller†

CNRS, LIPN – UMR 7030 Université Sorbonne Paris Nord  
<https://www.seiller.org>  
[seiller@lipn.fr](mailto:seiller@lipn.fr)

February 21, 2020

## Abstract

This paper presents a new abstract method for proving lower bounds in computational complexity. Based on the notion of topological entropy for dynamical systems, the method captures four previous lower bounds results from the literature in algebraic complexity [SY82, BO83, Cuc92, Mul99]. Among these results lies Mulmuley’s proof that “prams without bit operations” do not compute the `maxflow` problem in polylogarithmic time [Mul99], which was the best known lower bounds in the quest for a proof that  $\text{NC} \neq \text{P}_{\text{TIME}}$ . Inspired from a refinement of Steele and Yao’s lower bounds [SY82], due to Ben-Or [BO83], we strengthen Mulmuley’s result to a larger class of machines, showing that PRAMs over integer do not compute `maxflow` in polylogarithmic time.

---

\*Luc Pellissier was partially supported by the ANR projects ELICA and RAPIDO and the ANII project “Realizabilidad, forcing y computación cuántica”.

†Thomas Seiller was partially supported by the H2020 programme Marie Skłodowska-Curie Individual Fellowship (H2020-MSCA-IF-2014) project - ReACT and the CNRS INS2I JCJC grants BiGRE and LoBE.

# 1 Introduction

The NC vs PTIME question is one of the foremost open questions in computational complexity. In laymen’s terms, it asks whether a problem efficiently computable on a sequential machine can be computed substantially more efficiently on a parallel machine. It is well known that any problem in NC, i.e. that is computable in polylogarithmic time on a parallel machine (with a polynomial number of processors), belongs to PTIME, i.e. is computable in polynomial time on a sequential machine. The converse, however, is expected to be false. Indeed, although many problems in PTIME can be shown to be in NC, some of them seem to resist efficient parallelisation. In particular it is not known whether the `maxflow` problem, known to be PTIME-complete [GSS82], belongs to NC.

As part of the investigations on the NC vs PTIME question, a big step forward is due to K. Mulmuley. In 1999 [Mul99], he showed that a notion of machine introduced under the name “PRAMS without bit operations” does not compute `maxflow` in polylogarithmic time. This notion of machine, quite exotic at first sight, corresponds to an algebraic variant of PRAMS, where registers contain integers and individual processors are allowed to perform sums, subtractions and products of integers. It is argued by Mulmuley that this notion of machine provides an expressive model of computation, able to compute some non trivial problems in NC such as Neff’s algorithm for computing approximate roots of polynomials [Nef94]. Although Mulmuley’s result represents a big step forward in the quest for a proof that PTIME and NC are not equal, the result was not strengthened or reused in the last 20 years, and remained the strongest known lower bound result.

**Contributions.** The present work started from a reformulation of Mulmuley’s proof, based on the mathematical theory of dynamical systems. At first, this allowed us to abstract the proof method, and realise the proof technique is the same as the one of two previous lower bound results from the literature: lower bounds for algebraic computational trees by Steele and Yao [SY82], and Cucker’s proof that  $\text{NC}_{\mathbf{R}}$  is different from  $\text{PTIME}_{\mathbf{R}}$  [Cuc92], where  $\text{NC}_{\mathbf{R}}$  and  $\text{PTIME}_{\mathbf{R}}$  are the algebraic variants of NC and PTIME for machines computing over real numbers. In a second step, we were able to refine the proof method, following Ben Or’s improvement [BO83] over Steele and Yao’s result. Finally, applying this refined proof method to algebraic PRAMS, we strengthen Mulmuley’s result by showing that PRAMS over the integers do not compute `maxflow` in polylogarithmic time by proving it holds for a strictly stronger notion of machines. I.e. the result holds for PRAMS over  $\mathbf{Z}$ , that is algebraic parallel random access machines where registers contain integers and individual processors are allowed to compute euclidean division and square roots in addition to the basic operations (sums, subtraction and products) allowed in Mulmuley’s model.

In the following section, we start by exposing the mathematics at work in our proof method. Inspired by earlier work in semantics by the second author [Sei16, Sei17, Sei18, Sei19], it relies strongly on the theory of dynamical systems. The next section then explains the concept of *topological entropy*, an invariant of dynamical systems that quantifies the growth of the number of orbits. Our first contribution is to explain how this is used to provide upper bounds on the number of different subsets a given machine can distinguish in a given number of computation steps. These upper bounds are used in the literature (though not

presented this way) to prove lower bounds results [SY82, Cuc92, Mul99] using the Milnor-Thom theorem [Mil64, Tho65]. We then continue by explaining how this general method can be improved following a technique introduced by Ben-Or [BO83] to generalise Steele and Yao’s lower bounds. Finally, we show that this improved technique can be used to strengthen the result of Mulmuley, and prove the following theorem, which is the main result of this paper.

**Theorem 1.** *Let  $M$  be a PRAM over  $\mathbf{Z}$  with  $2^{O((\log N)^c)}$  processors, with  $N$  the length of the inputs and  $c$  any positive integer. Then  $M$  does not decide `maxflow` in  $O((\log N)^c)$  steps.*

This result improves on the previous strongest known lower bounds obtained by Mulmuley [Mul99], since Mulmuley showed the exact same statement for “PRAMS without bit operations”, that is PRAMS over  $\mathbf{Z}$  whose processors use only sums, subtractions and multiplications.

## 2 Programs as Dynamical systems

### 2.1 Abstract models of computation and graphings

We consider computations as a dynamical process, hence model them as a dynamical systems with two main components: a space  $\mathbf{X}$  that abstracts the notion of configuration space and a monoid acting on this space that represents the different operations allowed in the model of computation. Although the notion of *space* considered can vary (one could consider e.g. topological spaces, measure spaces, topological vector spaces), we restrict ourselves to topological spaces in this work.

**Definition 2.** An *abstract model of computation* (AMC) is defined as a monoid action  $\alpha : M \curvearrowright \mathbf{X}$ , i.e. a monoid morphism from  $M$  to the group of endomorphisms of  $\mathbf{X}$ .

The monoid  $M$  is often given with a set  $G$  of generators and a set of relations  $R$  which can be deduced from the image of  $G$  through  $\alpha$ . We denote such an AMC as  $\alpha : \langle G, R \rangle \curvearrowright \mathbf{X}$ .

Programs in an AMC  $\alpha : \langle G, R \rangle \curvearrowright \mathbf{X}$  is then defined as *graphings*, i.e. graphs whose vertices are subspaces of the space  $\mathbf{X}$  (representing sets of configurations on which the program act in the same way) and edges are labelled by elements of  $M\langle G, R \rangle$ , together with a global control state. More precisely, we use here the notion of *topological graphings*<sup>1</sup> [Sei17].

**Definition 3.** A graphing representative  $G$  w.r.t. a monoid action  $\alpha : M \curvearrowright \mathbf{X}$  is defined as a set of *edges*  $E^G$  and for each element  $e \in E^G$  a pair  $(S_e^G, m_e^G)$  of a subspace  $S_e^G$  of  $\mathbf{X}$  – the *source* of  $e$  – and an element  $m_e^G \in M$  – the *realiser* of  $e$ .

While graphing representatives are convenient to manipulate, they do provide too much information about the programs. Indeed, if one is to study programs as dynamical systems, the focus should be on the *dynamics*, i.e. on

<sup>1</sup>While “measured” graphings were already considered [Sei17], the definition adapts in a straightforward manner to allow for other notions such as graphings over topological vector spaces – which would be objects akin to the notion of quiver used in representation theory.

how the object acts on the underlying space. The following notion of *refinement* captures this idea that the same dynamics might be represented in many different ways in a graph-like manner.

**Definition 4** (Refinement). A graphing representative  $F$  is a refinement of a graphing representative  $G$ , noted  $F \leq G$ , if there exists a partition  $(E_e^F)_{e \in E^G}$  of  $E^F$  such that  $\forall e \in E^G$ :

$$(\cup_{f \in E_e^F} S_f^F) \Delta S_e^G = \emptyset; \quad \forall f \neq f' \in E_e^F, \quad S_f^F \Delta S_{f'}^F = \emptyset; \quad \forall f \in E_e^F, \quad m_f^F = m_e^G$$

This induces an equivalence relation defined as  $F \sim G \Leftrightarrow \exists H, H \leq F \wedge H \leq G$ .

The notion of *graphing* is therefore obtained by considering the quotient of the set of graphing representative w.r.t. refinement. More concretely, graphings are obtained by quotienting the set of graphing representatives w.r.t. the equivalence relation defined as “two graphings are equivalent if their action on the underlying space are equal”.

**Definition 5.** A *graphing* is an equivalence class of graphing representatives w.r.t. the equivalence relation induced by refinements.

**Definition 6.** Given an AMC  $\alpha : M \curvearrowright \mathbf{X}$ , an *abstract program*  $A$  is a graphing  $G^A$  w.r.t. the monoid action  $\alpha \times \mathfrak{S}_k \curvearrowright \mathbf{X} \times \mathbf{S}^A$ , where  $\mathbf{S}^A$  is a finite set of *control states* of cardinality  $k$  and  $\mathfrak{S}_k$  is the group of permutations of  $k$  elements.

Now, as a sanity check, we will show how the notion of graphing do capture the dynamics as expected. For this, we restrict to *deterministic graphings*, and show the notion relates to the usual notion of dynamical system.

**Definition 7.** A graphing representative  $G$  is deterministic if for all  $x \in \mathbf{X}$  there is at most one  $e \in E^G$  such that  $x \in S_e^G$ . A graphing is *deterministic* if its representatives are deterministic. An abstract program is *deterministic* if its underlying graphing is deterministic.

**Lemma 8.** *There is a one-to-one correspondence between deterministic graphings w.r.t. the monoid action  $M \curvearrowright \mathbf{X}$  and the set of partial dynamical systems  $f : \mathbf{X} \hookrightarrow \mathbf{X}$  whose graph is included in the partial order<sup>2</sup> and  $\{(x, y) \mid \exists m \in M, \alpha(m)(x) = y\}$ .*

## 2.2 Algebraic SRAMs

In this paper, we will consider algebraic parallel random access machines, that act not on strings of bits, but on integers. In order to define those properly, we first define the notion of (sequential) random access machine (SRAM) before considering their parallelisation.

An SRAM *command* is a pair  $(\ell, I)$  of a *line*  $\ell \in \mathbf{N}^*$  and an *instruction*  $I$  among the following, where  $i, j \in \mathbf{N}$ ,  $\star \in \{+, -, \times, /\}$ ,  $c \in \mathbf{Z}$  is a constant and  $\ell, \ell' \in \mathbf{N}^*$  are lines:

`skip;  $X_i := c$ ;  $X_i := X_j \star X_k$ ;  $X_i := X_j$ ;  $X_i := \#X_j$ ;  $\#X_i := X_j$ ; if  $X_i = 0$  goto  $\ell$  else  $\ell'$ ;`

<sup>2</sup>When  $\alpha$  is a group action acting by measure-preserving transformations, this is a *Borel equivalence relation*  $\mathcal{R}$ , and the condition stated here boils down to requiring that  $f$  belongs to the *full group* of  $\alpha$ .

A SRAM *machine*  $M$  is then a finite set of commands such that the set of lines is  $\{1, 2, \dots, |M|\}$ , with  $|M|$  the *length* of  $M$ . We will denote the commands in  $M$  by  $(i, \text{Inst}_M(i))$ , i.e.  $\text{Inst}_M(i)$  denotes the line  $i$  instruction. As the semantics are straightforward, we will only present their formalization as an AMC.

We now define the SRAM action. As we intend to consider PRAMs, we consider from the beginning the memory of a SRAM to be separated in two infinite blocks  $\mathbf{Z}^\omega$ , intended to represent both *shared* and a *private* memory cells. The underlying space is  $\mathbf{X} = \mathbf{Z}^{\mathbf{Z}} \cong \mathbf{Z}^\omega \times \mathbf{Z}^\omega$ . The set of generators is defined following the possible actions of an SRAM on the memory:  $\text{const}_i(c)$ ,  $\text{add}_i(j, k)$ ,  $\text{sub}_i(j, k)$ ,  $\text{mult}_i(j, k)$ ,  $\text{eucl}_i(j, k)$ ,  $\text{copy}(i, j)$ ,  $\text{copy}(\#i, j)$ ,  $\text{copy}(i, \#j)$ .

Each of the generator acts as follows, with  $\star \in \{+, -, \times\}$  and writing  $k//n$  the floor of  $k/n$  with the convention that  $k//n = 0$  when  $n = 0$ :

$$\begin{aligned}
\bullet \alpha(\text{const}_i(c))(\vec{x}) &= (\vec{x}\{x_i := c\}); & \bullet \alpha(\text{copy}(\#i, j))(\vec{x}) &= (\vec{x}\{x_i := x_{x_j}\}); \\
\bullet \alpha(\star_i(j, k))(\vec{x}) &= (\vec{x}\{x_i := x_j \star x_k\}); & \bullet \alpha(\text{copy}(i, \#j))(\vec{x}) &= (\vec{x}\{x_{x_i} := x_j\}); \\
\bullet \alpha(\text{copy}(i, j))(\vec{x}) &= (\vec{x}\{x_i := x_j\}); & \bullet \alpha(\text{eucl}_i(j, k))(\vec{x}) &= (\vec{x}\{x_i := x_j // x_k\}).
\end{aligned}$$

Machines in the SRAM model can be represented as graphings w.r.t. this action; intuitively the encoding works as follows. The notion of *control state* allows to represent the notion of *line* in the program. Then, the action just defined allows for the representation of all commands but the conditionals. The conditionals are represented as follows: depending on the value of  $X_i$  one wants to jumps either to the line  $\ell$  or to the line  $\ell'$ ; this is easily modelled by two different edges of respective sources  $\mathbb{H}(i) = \{\vec{x} \mid x_i = 0\}$  and  $\mathbb{H}(i)^c = \{\vec{x} \mid x_i \neq 0\}$ .

**Definition 9.** Let  $M$  be a SRAM machine. We define  $[M]$  as the graphing with set of control states  $\{0, 1, \dots, L, L+1\}$  where each line  $\ell$  defines:

- a single edge  $e$  of source  $\mathbf{X} \times \{\ell\}$  and realised by (in the following,  $\star \in \{+, -, \times\}$  and we write  $\ell++$  the map  $\ell \mapsto \ell + 1$ ):

$$\begin{aligned}
- (\text{Id}, \ell++) \text{ if } \text{Inst}_M(\ell) = \text{skip}; & & \mathbf{X}_i &:= \mathbf{X}_j; \\
- (\text{const}_i(c), \ell++) \text{ if } \text{Inst}_M(\ell) &= & - (\text{copy}(i, \#j), \ell++) \text{ if } \text{Inst}_M(\ell) &= \\
\mathbf{X}_i &:= c; & \mathbf{X}_i &:= \# \mathbf{X}_j; \\
- (\star_i(j, k), \ell++) \text{ if } \text{Inst}_M(\ell) &= & - (\text{copy}(\#i, j), \ell++) \text{ if } \text{Inst}_M(\ell) &= \\
\mathbf{X}_i &:= \mathbf{X}_j \star \mathbf{X}_k; & \# \mathbf{X}_i &:= \mathbf{X}_j. \\
- (\text{copy}(i, j), \ell++) \text{ if } \text{Inst}_M(\ell) &= & &
\end{aligned}$$

- an edge  $e$  of source  $\mathbb{H}(k) \times \{\ell\}$  realised by  $(\text{eucl}_i(j, k), \ell \mapsto \ell + 1)$  if  $\text{Inst}_M(\ell)$  is  $\mathbf{X}_i := \mathbf{X}_j / \mathbf{X}_k$ ;
- a pair of edges  $e, e^c$  of respective sources  $\mathbb{H}(i) \times \{\ell\}$  and  $\mathbb{H}(i)^c \times \{\ell\}$  and realised by respectively  $(\text{Id}, \ell \mapsto \ell^0)$  and  $(\text{Id}, \ell \mapsto \ell^1)$ , if the line is a conditional **if**  $\mathbf{X}_i = 0$  **goto**  $\ell^0$  **else**  $\ell^1$ .

We immediately define two other models of computations, derived from the algebraic SRAMs. First are the real-valued SRAMs defined in the same way than the integer-valued ones, but with underlying space  $\mathbf{X} = \mathbf{R}^{\mathbf{Z}}$  and the instructions are adapted accordingly:

- the division is the usual operation on the reals:  $\alpha(\text{euc1}_i(j, k))(\vec{x}) = (\vec{x}\{x_i := x_j/x_k\})$ ;
- the three copy operators are only effective on integers: for instance  $\alpha(\text{copy}(i, j))(\vec{x}) = \begin{cases} \vec{x}\{x_i := x_j\} & \text{if } i \text{ and } j \text{ are integers} \\ \vec{x} & \text{else} \end{cases}$

This AMC (of real-valued algebraic SRAMs) can be enriched by other operations, such as the square root. We say that a SRAM such that all the conditionals are of the form  $(\ell, \text{if } X_i = 0 \text{ goto } \ell^0 \text{ else } \ell^1)$  with  $\ell^0, \ell^1 > \ell$  is *monotonous* as its control flow only goes forward. A monotonous real-valued algebraic SRAM with square roots is called an *algebraic computational tree*: indeed, we recover in this way the notion defined by, e.g., Ben-Or [BO83].

### 2.3 The Crew

Based on the notion of SRAM, we are now able to consider their parallelisation, namely PRAMs. A PRAM  $M$  is given as a finite sequence of SRAM machines  $M_1, \dots, M_p$ , where  $p$  is the number of *processors* of  $M$ . Each processor  $M_i$  has access to its own, private, set of registers  $(X_k^i)_{k \geq 0}$  and a *shared memory* represented as a set of registers  $(X_k^0)_{k \geq 0}$ .

One has to deal with conflicts when several processors try to access the shared memory simultaneously. We here chose to work with the *Concurrent Read, Exclusive Write* (CREW) discipline: at a given step at which several processors try to write in the shared memory, only the processor with the smallest index will be allowed to do so. In order to model such parallel computations, we abstract the CREW at the level of monoids. For this, we suppose that we have two monoid actions  $M\langle G, R \rangle \curvearrowright \mathbf{X} \times \mathbf{Y}$  and  $M\langle H, Q \rangle \curvearrowright \mathbf{X} \times \mathbf{Z}$ , where  $\mathbf{X}$  represents the shared memory. We then consider the subset  $\# \subset G \times H$  of pairs of generators that potentially conflict with one another – the conflict relation.

**Definition 10** (Conflicted sum). Let  $M\langle G, R \rangle, M\langle G', R' \rangle$  be two monoids and  $\# \subseteq G \times G'$ . The *conflicted sum* of  $M\langle G, R \rangle$  and  $M\langle G', R' \rangle$  over  $\#$ , noted  $M\langle G, R \rangle *_{\#} M\langle G', R' \rangle$ , is defined as the monoid with generators  $(\{1\} \times G) \cup (\{2\} \times G')$  and relations

$$(\{1\} \times R) \cup (\{2\} \times R') \cup \{((1, g)(2, g'), (2, g')(1, g)) \mid (g, g') \notin \#\} \cup \{(1, e)\} \cup \{(1, e')\}$$

where  $\mathbf{1}, e, e'$  are the neutrals of  $M\langle G, R \rangle *_{\#} M\langle G', R' \rangle, M\langle G, R \rangle$  and  $M\langle G', R' \rangle$  respectively.

In the particular case where  $\# = (G \times H') \cup (H \times G')$ , with  $H, H'$  respectively subsets of  $G$  and  $G'$ , we will write the sum  $M\langle G, R \rangle_{H *_{H'}} M\langle G', R' \rangle$ .

*Remark.* When the conflict relation  $\#$  is empty, this defines the usual direct product of monoids. This corresponds to the case in which no conflicts can arise w.r.t. the shared memory. In other words, the direct product of monoids corresponds to the parallelisation of processes *without shared memory*.

Dually, when the conflict relation is full ( $\# = G \times G'$ ), this defines the free product of the monoids, i.e. the case in which all instructions interact with the shared memory and therefore no real parallelisation is possible.

**Definition 11.** Let  $\alpha : M \curvearrowright \mathbf{X} \times \mathbf{Y}$  be a monoid action. We say that an element  $m \in M$  is *central relatively to  $\alpha$*  (or just *central*) if the action of  $m$

commutes with the first projection  $\pi_X : \mathbf{X} \times \mathbf{Y} \rightarrow \mathbf{X}$ , i.e.<sup>3</sup>  $\alpha(m); \pi_X = \alpha(m)$ ; in other words  $m$  acts as the identity on  $\mathbf{X}$ .

Intuitively, central elements are those that will not affect the shared memory. As such, only *non-central elements* need to be dealt with care when the processes are put in parallel.

**Definition 12.** Let  $M\langle G, R \rangle \curvearrowright \mathbf{X} \times \mathbf{Y}$  be an AMC. We note  $Z_\alpha$  the set of central elements and  $\bar{Z}_\alpha(G)$  the set  $\{m \in G \mid m \notin Z_\alpha\}$ .

**Definition 13** (The CREW of AMCs). Let  $\alpha : M\langle G, R \rangle \curvearrowright \mathbf{X} \times \mathbf{Y}$  and  $\beta : M\langle H, Q \rangle \curvearrowright \mathbf{X} \times \mathbf{Z}$  be AMCs. We define the AMC  $\text{CREW}(\alpha, \beta) : M\langle G, R \rangle_{\bar{Z}_\alpha(G)} *_{\bar{Z}_\beta(G')} M\langle G', R' \rangle \curvearrowright \mathbf{X} \times \mathbf{Y} \times \mathbf{Z}$  by letting  $\text{CREW}(\alpha, \beta)(m, m') = \alpha(m) * \beta(m')$  on elements of  $G \times G'$ , where:

$$\alpha(m) * \beta(m') = \begin{cases} \Delta; [\alpha(m); \pi_Y, \beta(m')] & \text{if } m \notin \bar{Z}_\alpha(G), m' \in \bar{Z}_\beta(G'), \\ \Delta; [\alpha(m), \beta(m'); \pi_Z] & \text{otherwise,} \end{cases}$$

with  $\Delta : (x, y, z) \mapsto (x, y, x, z) : \mathbf{X} \times \mathbf{Y} \times \mathbf{Z} \rightarrow \mathbf{X} \times \mathbf{Y} \times \mathbf{X} \times \mathbf{Z}$ .

We can now define AMC of PRAMS and thus the interpretations of PRAMS as abstract programs. For each integer  $p$ , we define the AMC  $\text{CREW}^p(\alpha)$  for  $\alpha$  is the AMC for SRAMS defined in the previous section. This allows the consideration of up to  $p$  parallel SRAMS. The interpretation of such a SRAM with  $p$  processors is then defined by considering a set of states equal to  $L_1 \times L_2 \times \dots \times L_p$  where for all  $i$  the set  $L_i$  is the set of labels of the  $i$ -th processor.

Now, to deal with arbitrary large PRAMS, i.e. with arbitrarily large number of processors, one considers the following AMC defined as a *direct limit*.

**Definition 14** (The AMC of PRAMS). Let  $\alpha : M \curvearrowright \mathbf{X} \times \mathbf{X}$  be the SRAM AMC. The AMC of PRAMS is defined as  $\varinjlim \text{CREW}^k(\alpha)$ , where  $\text{CREW}^{k-1}(\alpha)$  is identified with a restriction of  $\text{CREW}^k(\alpha)$  through  $\text{CREW}^{k-1}(\alpha)(m_1, \dots, m_{k-1}) \mapsto \text{CREW}^k(\alpha)(m_1, \dots, m_{k-1}, 1)$ .

Remark that the underlying space of the PRAM AMC is defined as the union  $\bigcup_{n \in \omega} \mathbf{Z}^\omega \times (\mathbf{Z}^\omega)^n$  which we will write  $\mathbf{Z}^\omega \times (\mathbf{Z}^\omega)^\omega$ .

## 3 Entropy and Cells

### 3.1 Topological Entropy

Topological Entropy is a standard invariant of dynamical system. It is a value representing the average exponential growth rate of the number of orbit segments distinguishable with a finite (but arbitrarily fine) precision. The definition is based on the notion of open covers.

**Definition 15** (Open covers). Given a topological space  $\mathbf{X}$ , an *open cover* of  $\mathbf{X}$  is a family  $\mathcal{U} = (U_i)_{i \in I}$  of open subsets of  $\mathbf{X}$  such that  $\bigcup_{i \in I} U_i = \mathbf{X}$ . A finite cover  $\mathcal{U}$  is a cover whose indexing set is finite. A *subcover* of a cover  $\mathcal{U} = (U_i)_{i \in I}$  is a sub-family  $\mathcal{S} = (U_j)_{j \in J}$  for  $J \subseteq I$  such that  $\mathcal{S}$  is a cover, i.e. such that  $\bigcup_{j \in J} U_j = \mathbf{X}$ . We will denote by  $\text{Cov}(\mathbf{X})$  (resp.  $\text{FCov}(\mathbf{X})$ ) the set of all open covers (resp. all finite open covers) of the space  $\mathbf{X}$ .

<sup>3</sup>Here and in the following, we denote by  $;$  the sequential composition of functions. I.e.  $f;g$  denotes what is usually written  $g \circ f$ .



**Definition 16.** An open cover  $\mathcal{U} = (U_i)_{i \in I}$ , together with a continuous function  $f : \mathbf{X} \rightarrow \mathbf{X}$ , defines the inverse image open cover  $f^{-1}(\mathcal{U}) = (f^{-1}(U_i))_{i \in I}$ . Given two open covers  $\mathcal{U} = (U_i)_{i \in I}$  and  $\mathcal{V} = (V_j)_{j \in J}$ , we define their join  $\mathcal{U} \vee \mathcal{V}$  as the family  $(U_i \cap V_j)_{(i,j) \in I \times J}$ .

*Remark.* If  $\mathcal{U}, \mathcal{V}$  are finite,  $f^{-1}(\mathcal{U})$  and  $\mathcal{U} \vee \mathcal{V}$  are finite as well.

Traditionally [AKM65], entropy is defined for continuous maps on a compact set. However, a generalisation of entropy to non-compact sets can easily be defined by restricting the usual definition to *finite* covers<sup>4</sup>. This is the definition we will use here.

**Definition 17.** Let  $\mathbf{X}$  be a topological space, and  $\mathcal{U} = (U_i)_{i \in I}$  be a finite cover of  $\mathbf{X}$ . We define the quantity  $H_{\mathbf{X}}^0(\mathcal{U}) = \min\{\log_2(\text{Card}(J)) \mid J \subset I, \cup_{j \in J} U_j = \mathbf{X}\}$ .

In other words, if  $k$  is the cardinality of the smallest subcover of  $\mathcal{U}$ ,  $H^0(\mathcal{U}) = \log_2(k)$ .

**Definition 18.** Let  $\mathbf{X}$  be a topological space and  $f : \mathbf{X} \rightarrow \mathbf{X}$  be a continuous map. For any finite open cover  $\mathcal{U}$  of  $\mathbf{X}$ , define  $H_{\mathbf{X}}^k(f, \mathcal{U}) = \frac{1}{k} H_{\mathbf{X}}^0(\mathcal{U} \vee f^{-1}(\mathcal{U}) \vee \dots \vee f^{-(k-1)}(\mathcal{U}))$ .

One can show that the limit  $\lim_{n \rightarrow \infty} H_{\mathbf{X}}^n(f, \mathcal{U})$  exists and is finite; it will be noted  $h(f, \mathcal{U})$ . The topological entropy of  $f$  is then defined as the supremum of these values, when  $\mathcal{U}$  ranges over the set of all finite covers  $\text{FCov}(\mathbf{X})$ .

**Definition 19.** Let  $\mathbf{X}$  be a topological space and  $f : \mathbf{X} \rightarrow \mathbf{X}$  be a continuous map. The *topological entropy* of  $f$  is defined as  $h(f) = \sup_{\mathcal{U} \in \text{FCov}(\mathbf{X})} h(f, \mathcal{U})$ .

## 3.2 Graphings and Entropy

We now need to define the entropy of *deterministic graphings*. As mentioned briefly already, deterministic graphings on a space  $\mathbf{X}$  are in one-to-one correspondence with partial dynamical systems on  $\mathbf{X}$ . Thus, we only need to extend the notion of entropy to partial maps, and we can then define the entropy of a graphing  $G$  as the entropy of its corresponding map  $[G]$ .

Given a finite cover  $\mathcal{U}$ , the only issue with partial continuous maps is that  $f^{-1}(\mathcal{U})$  is not in general a cover. Indeed,  $\{f^{-1}(U) \mid U \in \mathcal{U}\}$  is a family of open sets by continuity of  $f$  but the union  $\cup_{U \in \mathcal{U}} f^{-1}(U)$  is a strict subspace of  $\mathbf{X}$  (namely, the domain of  $f$ ). It turns out the solution to this problem is quite simple: we notice that  $f^{-1}(\mathcal{U})$  is a cover of  $f^{-1}(\mathbf{X})$  and now work with covers of subspaces of  $\mathbf{X}$ . Indeed,  $\mathcal{U} \vee f^{-1}(\mathcal{U})$  is itself a cover of  $f^{-1}(\mathbf{X})$  and therefore the quantity  $H_{\mathbf{X}}^2(f, \mathcal{U})$  can be defined as  $(1/2)H_{f^{-1}(\mathbf{X})}^0(\mathcal{U} \vee f^{-1}(\mathcal{U}))$ .

We now generalise this definition to arbitrary iterations of  $f$  by extending Definitions 18 and 19 to partial maps as follows.

**Definition 20.** Let  $\mathbf{X}$  be a topological space and  $f : \mathbf{X} \rightarrow \mathbf{X}$  be a continuous partial map. For any finite open cover  $\mathcal{U}$  of  $\mathbf{X}$ , we define  $H_{\mathbf{X}}^k(f, \mathcal{U}) = \frac{1}{k} H_{f^{-k+1}(\mathbf{X})}^0(\mathcal{U} \vee f^{-1}(\mathcal{U}) \vee \dots \vee f^{-(k-1)}(\mathcal{U}))$ . The *entropy* of  $f$  is then defined as  $h(f) = \sup_{\mathcal{U} \in \text{FCov}(\mathbf{X})} h(f, \mathcal{U})$ , where  $h(f, \mathcal{U})$  is again defined as the limit  $\lim_{n \rightarrow \infty} H_{\mathbf{X}}^n(f, \mathcal{U})$ .

<sup>4</sup>This is discussed by Hofer [Hof75] together with another generalisation based on the Stone-Ćech compactification of the underlying space.

We now consider the special case of a graphing  $G$  with set of control states  $S^G$ . For an intuitive understanding, one can think of  $G$  as the representation of a PRAM machine. We focus on the specific open cover indexed by the set of control states, i.e.  $\mathcal{S} = (\mathbf{X} \times \{s\}_{s \in S^G})$ , and call it *the states cover*. We will now show how the partial entropy  $H^k(G, \mathcal{S})$  is related to the set of *admissible sequence of states*. Let us define those first.

**Definition 21.** Let  $G$  be a graphing, with set of control states  $S^G$ . An admissible sequence of states is a sequence  $\mathbf{s} = s_1 s_2 \dots s_n$  of elements of  $S^G$  such that for all  $i \in \{1, 2, \dots, n-1\}$  there exists a subset  $C$  of  $\mathbf{X}$  – i.e. a set of configurations – such that  $G$  contains an edge from  $C \times \{s_i\}$  to a subspace of  $\mathbf{X} \times \{s_{i+1}\}$ .

*Example 22.* As an example, let us consider the very simple graphing with four control states  $a, b, c, d$  and edges from  $\mathbf{X} \times \{a\}$  to  $\mathbf{X} \times \{b\}$ , from  $\mathbf{X} \times \{b\}$  to  $\mathbf{X} \times \{c\}$ , from  $\mathbf{X} \times \{c\}$  to  $\mathbf{X} \times \{b\}$  and from  $\mathbf{X} \times \{c\}$  to  $\mathbf{X} \times \{d\}$ . Then the sequences  $abcd$  and  $abcbbc$  are admissible, but the sequences  $aba$ ,  $abcd$ , and  $abca$  are not.

**Lemma 23.** Let  $G$  be a graphing, and  $\mathcal{S}$  its states cover. Then for all integer  $k$ , the set  $\text{Adm}_k(G)$  of admissible sequences of states of length  $k > 1$  is of cardinality  $2^{k \cdot H^k(G, \mathcal{S})}$ .

A tractable bound on the number of admissible sequences of states can be obtained by noticing that the sequence  $H^k(G, \mathcal{S})$  is *sub-additive*, i.e.  $H^{k+k'}(G, \mathcal{S}) \leq H^k(G, \mathcal{S}) + H^{k'}(G, \mathcal{S})$ . A consequence of this is that  $H^k(G, \mathcal{S}) \leq kH^1(G, \mathcal{S})$ . Thus the number of admissible sequences of states of length  $k$  is bounded by  $2^{k^2 H^1(G, \mathcal{S})}$ . We now study how the cardinality of admissible sequences can be related to the entropy of  $G$ . This is deduced from Lemma 23 and the following general result (which does not depend on the chosen cover).

**Lemma 24.** For all  $\epsilon > 0$  and all cover  $\mathcal{U}$ , there exists a natural number  $N$  such that  $\forall k \geq N$ ,  $H^k(G, \mathcal{U}) < h([G]) + \epsilon$ .

The two previous lemmas combine to give the following.

**Lemma 25.** Let  $G$  be a graphing. Then  $\text{Card}(\text{Adm}_k(G)) = O(2^{k \cdot h([G])})$  as  $k \rightarrow \infty$ .

### 3.3 Cells Decomposition

Now, let  $G$  be a deterministic graphing with its state cover  $\mathcal{S}$ . We fix  $k > 2$  and consider the partition  $(C[\mathbf{s}])_{\mathbf{s} \in \text{Adm}_k(G)}$  of the space  $[G]^{-k+1}(\mathbf{X})$ , where the sets  $C[\mathbf{s}] = C[(s_1 s_2 \dots s_{k-1}, s_k)]$  are defined inductively as follow:

- $C[s_1, s_2]$  is the set  $\{x \in \mathbf{X} \mid [G](x, s_1) \in X \times \{s_2\}\}$ ;
- $C[(s_1 s_2 \dots s_{k-1}, s_k)]$  is the set  $\{x \in \mathbf{X} \mid \forall i \in \{2, \dots, k\}, [G]^{i-1}(x, s_1) \in \mathbf{X} \times \{s_i\}\}$ .

This decomposition splits the set of initial configurations into cells satisfying the following property: *for any two initial configurations contained in the same cell  $C[\mathbf{s}]$ , the  $k$ -th first iterations of  $G$  go through the same admissible sequence of states  $\mathbf{s}$ .*

**Definition 26.** Let  $G$  be a deterministic graphing, with its state cover  $\mathcal{S}$ . Given an integer  $k$ , we define the  $k$ -th cell decomposition of  $\mathbf{X}$  along  $G$  as the partition  $\{C[\mathbf{s}] \mid \mathbf{s} \in \text{Adm}_k(G)\}$ .

Then Lemma 23 provides a bound on the cardinality of the  $k$ -th cell decomposition. Using the results in the previous section, we can then obtain the following proposition.

**Proposition 27.** *Let  $G$  be a deterministic graphing, with entropy  $h(G)$ . The cardinality of the  $k$ -th cell decomposition of  $\mathbf{X}$  w.r.t.  $G$ , as a function  $c(k)$  of  $k$ , is asymptotically bounded by  $g(k) = 2^{k \cdot h(G)}$ , i.e.  $c(k) = O(g(k))$ .*

We also state another bound on the number of cells of the  $k$ -th cell decomposition, based on the state cover entropy, i.e. the entropy with respect to the state cover rather than the usual entropy which takes the supremum of cover entropies when the cover ranges over all finite covers of the space. This result is a simple consequence of Theorem 23.

**Proposition 28.** *Let  $G$  be a deterministic graphing. We consider the state cover entropy  $h_0([G]) = \lim_{n \rightarrow \infty} H_{\mathbf{X}}^n([G], \mathcal{S})$  where  $\mathbf{S}$  is the state cover. The cardinality of the  $k$ -th cell decomposition of  $\mathbf{X}$  w.r.t.  $G$ , as a function  $c(k)$  of  $k$ , is asymptotically bounded by  $g(k) = 2^{k \cdot h_0([G])}$ , i.e.  $c(k) = O(g(k))$ .*

## 4 Algebraic Computation Trees and Ben-Or's technique

### 4.1 Lower Bounds through the Milnor-Thom theorem

The results stated in the last section can be used to prove lower bounds in several models. These results rely on two ingredients: the above bounds on the cardinality of the  $k$ -th cell decomposition, and the Milnor-Thom theorem.

The Milnor-Thom theorem, which was proven independently by Milnor [Mil64] and Thom [Tho65], states bounds on the sum of the Betti numbers (i.e. the rank of the homology groups) of an algebraic variety. This theorem provides bounds on the number of connected components (i.e. the 0-th Betti number  $\beta_0(V)$ ) of a semi-algebraic variety  $V$ . We here use the statement of the Milnor-Thom theorem as given by Ben-Or [BO83, Theorem 2].

**Theorem 29.** *Let  $V \subseteq \mathbf{R}^n$  be a set defined by polynomial in-equations ( $n, m, h \in \mathbf{N}$ ):*

$$\{q_i(\vec{x}) = 0 \mid 0 \leq i \leq m\} \cup \{p_i(\vec{x}) > 0 \mid 0 \leq i \leq s\} \cup \{p_i(\vec{x}) \leq 0 \mid s+1 \leq i \leq h\}.$$

*Then  $\beta_0(V)$  is at most  $d(2d-1)^{n+h-1}$ , where  $d = \max\{2, \deg(q_i), \deg(p_j)\}$ .*

The lower bounds proofs then proceed by the following proof strategy:

1. consider an algebraic model of computation, and define the corresponding AMC;
2. show that the cells in the  $k$ -th cell decomposition are semi-algebraic sets defined by systems of equations  $E$  with explicit *upper bounds* on the number of equations and the degrees of the polynomials;

3. bound the number of connected components of each cell by the Milnor-Thom theorem;
4. given an algebraic problem (e.g. a subset of  $\mathbf{R}^k$ ), deduce lower bounds on the length of the computations deciding that problem based on its number of connected components.

Among the lower bound proofs using this proof strategy, we point out Steele and Yao lower bounds on algebraic decision trees [SY82], Cucker’s proof that  $\text{NC}_{\mathbf{R}} \subsetneq \text{PTIME}_{\mathbf{R}}$  [Cuc92], as well as Mulmuley’s proof of lower bounds on “PRAMS without bit operations” [Mul99]. These results, which do not use the notion of entropy, and do not cite each other, are thus for the first time related by our abstract point of view. For more details, we point the interested reader to an unpublished research report [PS18b] or the long version of this work [PS18a].

We will now explain how this method can be refined following Ben-Or’s proof of lower bounds for algebraic computational trees. Indeed, while Cucker’s result [Cuc92] and Mulmuley’s [Mul99] were not later improved upon<sup>5</sup>, Steele and Yao’s lower bounds were extended by Ben-Or [BO83] to encompass algebraic computational trees with sums, subtractions, products, divisions and square roots. The technique of Ben-Or improves on Steele and Yao in that it provides a method to deal with divisions and square roots. We here abstract this method by considering *k-th entropic co-trees* which are a refinement of the *k-th* cell decomposition. This allows us to recover Ben-Or’s result and, most importantly, to strengthen Mulmuley’s result by allowing the machines considered to use divisions and square roots.

## 4.2 Entropic co-trees

The principle underlying the improvement of Ben-Or on Steele and Yao consists in adding additional variables to avoid using the square root or division, obtaining in this way a system of polynomial equations instead of a single equation for a given cell in the *k-th* cell decomposition. For instance, instead of writing the equation  $p/q < 0$ , one defines a fresh variable  $r$  and considers the system  $\{p = qr; r < 0\}$ .

To adapt it to graphings, we consider the notion of *entropic co-tree* of a graphing that generalises the *k-th* cell decomposition to account for the instructions used at each step of the computation.

**Definition 30** (*k-th entropic co-tree*). Consider a deterministic graphing representative  $T$ , and fix an element  $\top$  of the set of control states. We can define the *k-th* entropic co-tree of  $T$  along  $\top$  and the state cover inductively:

- $k = 0$ , the co-tree  $\text{coT}_0(T)$  is simply the root  $n^\epsilon = \mathbf{R}^n \times \{\top\}$ ;
- $k = 1$ , one considers the preimage of  $n^\epsilon$  through  $T$ , i.e.  $T^{-1}(\mathbf{R}^n \times \{\top\})$  the set of all non-empty sets  $\alpha(m_e)^{-1}(\mathbf{R}^n \times \{\top\})$  and intersects it pairwise with the state cover, leading to a finite family (of cardinality bounded by the number of states multiplied by the number of edges of  $T$ )  $(n_e^i)_i$  defined as  $n^i = T^{-1}(n^\epsilon) \cap \mathbf{R}^n \times \{i\}$ . The first entropic co-tree  $\text{coT}_1(T)$  of  $T$  is then the tree defined by linking each  $n_e^i$  to  $n^\epsilon$  with an edge labelled by  $m_e$ ;

---

<sup>5</sup>Although one may argue that Cucker’s result did not call for improvement, this is not true of Mulmuley’s.

- $k + 1$ , suppose defined the  $k$ -th entropic co-tree of  $T$ , defined as a family of elements  $n_{\mathbf{e}}^\pi$  where  $\pi$  is a finite sequence of states of length at most  $k$  and  $\mathbf{e}$  a sequence of edges of  $T$  of the same length, and where  $n_{\mathbf{e}}^\pi$  and  $n_{\mathbf{e}'}^{\pi'}$  are linked by an edge labelled  $f$  if and only if  $\pi' = \pi.s$  and  $\mathbf{e}' = f.\mathbf{e}$  where  $s$  is a state and  $f$  an edge of  $T$ . We consider the subset of elements  $n_{\mathbf{e}}^\pi$ , where  $\pi$  is exactly of length  $k$ , and for each such element we define new vertices  $n_{\mathbf{e},\mathbf{e}'}^{\pi,s}$  defined as  $\alpha(m_e)^{-1}(n_{\mathbf{e}}^\pi) \cap \mathbf{R}^n \times \{s\}$  when it is non-empty. The  $k + 1$ -th entropic co-tree  $\text{COT}_{k+1}(T)$  is defined by extending the  $k$ -th entropic co-tree  $\text{COT}_k(T)$ , adding the vertices  $n_{\mathbf{e},\mathbf{e}'}^{\pi,s}$  and linking them to  $n_{\mathbf{e}}^\pi$  with an edge labelled by  $e$ .

This definition formalises a notion that appears more or less clearly in the work of Steele and Yao, and of Ben-Or, as well as in the proof by Mulmuley. The vertices for paths of length  $k$  in the  $k$ -th co-tree corresponds to the  $k$ -th cell decomposition, and the corresponding path defines the polynomials describing the semi-algebraic set decided by a computational tree. While in Steele and Yao and Mulmuley's proofs, one obtain directly a polynomial for each cell, we here need to construct a system of equations for each branch of the co-tree. Before explaining how this can be done, we first state bounds on the size of entropic co-trees.

**Proposition 31.** *Let  $G$  be a deterministic graphing with a finite set of edges  $E$ , and  $\text{Seq}_k(E)$  the set of length  $k$  sequences of edges in  $G$ . We consider the state cover entropy  $h_0([G]) = \lim_{n \rightarrow \infty} H_{\mathbf{X}}^n([G], \mathcal{S})$  where  $\mathbf{S}$  is the state cover. The cardinality of the length  $k$  vertices of the entropic co-tree of  $G$ , as a function  $c(k)$  of  $k$ , is asymptotically bounded by  $g(k) = \text{Card}(\text{Seq}_k(E)) \cdot 2^{k \cdot h_0([G])}$ , which is itself bounded by  $2^{\text{Card}(E)} \cdot 2^{k \cdot h_0([G])}$ .*

### 4.3 Abstracting Ben-Or's technique

We now use Ben-Or proof technique to obtain a bound on the number of connected components of the subsets  $W \subseteq \mathbf{R}^n$  whose membership problem is computed by a graphing in less than a given number of iterations.

This theorem specialises to the original theorem by Ben-Or relating the number of connected components of a set  $W$  and the depth of the algebraic computational trees that compute the membership problem for  $W$ .

**Theorem 32.** *Let  $G$  be a computational graphing representative with edges realised only by generators of the AMC of algebraic computational trees, and  $\text{Seq}_k(E)$  the set of length  $k$  sequences of edges in  $G$ . Suppose  $G$  computes the membership problem for  $W \subseteq \mathbf{R}^n$  in  $k$  steps, i.e. for each element of  $\mathbf{R}^n$ ,  $\pi_{\mathbf{S}}(G^k(x)) = \top$  if and only if  $x \in W$ . Then  $W$  is a semi-algebraic set defined by at most  $\text{Card}(\text{Seq}_k(E)) \cdot 2^{h_0([G])}$  systems of  $k$  equations of degree at most 2 and involving at most  $k + n$  variables.*

The proof of this theorem is long but simple to understand. We define, for each vertex of the  $k$ -th entropic co-tree, a system of algebraic equations (each of degree at most 2). The system is defined by induction on  $k$ , and uses the information of the specific instruction used to extend the sequence indexing the vertex at each step. Division for instance is dealt with following Ben-Or's method, by introducing a fresh variable and writing down the two equations written as the beginning of Section 4.2.

*Remark.* The total degree<sup>6</sup> of the system is bounded by  $2k\text{Card}(\text{Seq}_k(E)).2^{h_0(|G|)}$ .

This theorem extends to the case of general computational graphings by considering the *algebraic degree* of the graphing.

**Definition 33** (Algebraic degree). The algebraic degree of an element of the AMC is the minimal number of generators needed to express it. The algebraic degree of a graphing is the maximum of the algebraic degrees of the realisers of its edges.

If an edge is realised by an element  $m$  of algebraic degree  $D$ , the method above applies by introducing the  $D$  new equations corresponding to the  $D$  generators used to define  $m$ . The following general result is then obtained by applying the Milnor-Thom theorem on the obtained systems of equations to bound the number of connected components of each cell.

**Theorem 34.** *Let  $G$  be a computational graphing representative,  $\text{Seq}_k(E)$  the set of length  $k$  sequences of edges in  $G$ , and  $D$  its algebraic degree. Suppose  $G$  computes the membership problem for  $W \subseteq \mathbf{R}^n$  in  $k$  steps, i.e. for each element of  $\mathbf{R}^n$ ,  $\pi_{\mathbf{S}}(G^k(x)) = \top$  if and only if  $x \in W$ . Then  $W$  has at most  $\text{Card}(\text{Seq}_k(E)).2^{h_0(|G|)+1}3^{2kD+n+1}$  connected components.*

**Corollary 35** ([BO83, Theorem 5]). *Let  $W \subseteq \mathbf{R}^n$  be any set, and let  $N$  be the maximum of the number of connected components of  $W$  and  $\mathbf{R}^n \setminus W$ . An algebraic computation tree computing the membership problem for  $W$  has height  $\Omega(\log N)$ .*

Finally, let us make the following remark, which will be essential in the next sections.

*Remark.* In the case of algebraic PRAMS discussed in the next sections, the  $k$ -th entropic co-tree  $\text{COT}_k(T)[M]$  of a machine  $M$  defines an algebraic computational tree which follows the  $k$ -th first steps of computation of  $M$ . I.e. the algebraic computational tree  $\text{COT}_k(T)[M]$  approximate the computation of  $M$  in such a way that  $M$  and  $\text{COT}_k(T)[M]$  behave in the exact same manner in the first  $k$  steps.

## 5 Algebraic surfaces for machines and problems

We now encode a specific decision problem and the run of a PRAM as two specific subsets of the same space and show that no short run of the machine can define the set of all instances of the decision problem.

### 5.1 Geometric Interpretation of Optimization Problems

Let  $\mathcal{P}_{\text{opt}}$  be an optimization problem on  $\mathbf{R}^d$ . Solving  $\mathcal{P}_{\text{opt}}$  on an instance  $t$  amounts to optimizing a function  $f_t(\cdot)$  over a space of parameters. We note  $\text{Max}\mathcal{P}_{\text{opt}}(t)$  this optimal value. An affine function  $\text{Param} : [p; q] \rightarrow \mathbf{R}^d$  is called a *parametrization* of  $\mathcal{P}_{\text{opt}}$ . Such a parametrization defines naturally a decision problem  $\mathcal{P}_{\text{dec}}$ : for all  $(x, y, z) \in \mathbf{Z}^3$ ,  $(x, y, z) \in \mathcal{P}_{\text{dec}}$  iff  $z > 0$ ,  $x/z \in [p; q]$  and  $y/z \leq \text{Max}\mathcal{P}_{\text{opt}} \circ \text{Param}(x/z)$ .

<sup>6</sup>cf. Section 5.2 for the definition.

In order to study the geometry of  $\mathcal{P}_{\text{dec}}$  in a way that makes its connection with  $\mathcal{P}_{\text{opt}}$  clear, we consider the ambient space to be  $\mathbf{R}^3$ , and we define the *ray*  $[p]$  of a point  $p$  as the half-line starting at the origin and containing  $p$ . The projection  $\Pi(p)$  of a point  $p$  on a plane is the intersection of  $[p]$  and the affine plane  $\mathcal{A}_1$  of equation  $z = 1$ . For any point  $p \in \mathcal{A}_1$ , and all  $p_1 \in [p]$ ,  $\Pi(p_1) = p$ . It is clear that for  $(p, p', q) \in \mathbf{Z}^2 \times \mathbf{N}^+$ ,  $\Pi((p, p', q)) = (p/q, p'/q, 1)$ .

The *cone*  $[C]$  of a curve  $C$  is the set of rays of points of the curve. The projection  $\Pi(C)$  of a surface or a curve  $C$  is the set of projections of points in  $C$ . We note  $\text{Front}$  the frontier set  $\text{Front} = \{(x, y, 1) \in \mathbf{R}^3 \mid y = \text{Max}\mathcal{P}_{\text{opt}} \circ \text{Param}(x)\}$ , and we remark that  $[\text{Front}] = \{(x, y, z) \in \mathbf{R}^2 \times \mathbf{R}^+ \mid y/z = \text{Max}\mathcal{P}_{\text{opt}} \circ \text{Param}(x/z)\}$ .

A machine  $M$  decides the problem  $\mathcal{P}_{\text{dec}}$  in  $k$  steps if the partition of accepting cells in  $\mathbf{Z}^3$  induced by the machine – i.e. the  $k$ -th cell decomposition – is finer than the one defined by the problem's frontier  $[\text{Front}]$  (which is defined by the equation  $y/z \leq \text{Max}\mathcal{P}_{\text{opt}} \circ \text{Param}(x/z)$ ).

**Parametric Complexity.** We now further restrict the class of problems we are interested in: we will only consider  $\mathcal{P}_{\text{opt}}$  such that  $\text{Front}$  is simple enough. Precisely:

**Definition 36.** We say that  $\text{Param}$  is an *affine parametrization* of  $\mathcal{P}_{\text{opt}}$  if  $\text{Max}\mathcal{P}_{\text{opt}} \circ \text{Param}$  is convex, piecewise linear with breakpoints  $\lambda_1 < \dots < \lambda_\rho$ , and such that all  $(\lambda_i)_i$  and  $(\text{Max}\mathcal{P}_{\text{opt}} \circ \text{Param}(\lambda_i))_i$  are rational. The *parametric complexity*  $\rho(\text{Param})$  is the number of breakpoints  $\rho$ . The *bitsize* of the parametrization is the maximum of the bitsizes of the numerators and denominators of the coordinates of the breakpoints of  $\text{Max}\mathcal{P}_{\text{opt}} \circ \text{Param}$ .

An optimization problem admitting an affine parametrization of complexity  $\rho$  is thus represented by a quite simple surface  $[\text{Front}]$ : the cone of the graph of a piecewise affine function, constituted of  $\rho$  segments. We call such a surface is a  $\rho$ -*fan* and define its bitsize as  $\beta$  if all its breakpoints are rational and the bitsize of their coordinates is less than  $\beta$ .

The restriction to such optimization problems seems quite dramatic when understood geometrically. Nonetheless,  $\text{maxflow}$  admits such a parametrization.

**Theorem 37** (Murty [Mur80], Carstensen [Car83]). *There exists an affine parametrization of bitsize  $O(n^2)$  and complexity  $2^{\Omega(n)}$  of the  $\text{maxflow}$  problem for directed and undirected networks, where  $n$  is the number of nodes in the network.*

## 5.2 Surfaces from a PRAM

Let  $M$  be an integer-valued PRAM. We can associate to it a real-valued PRAM  $M'$  such that  $M$  and  $M'$  accept the same (integer) values, and the ratio between the running time of the two machines is a constant. Indeed,

**Proposition 38.** *Euclidian division can be computed by a constant time real-valued PRAM.*

Suppose now that  $M$  has, as a graphing, a finite set of edges  $\text{Card}(E)$ .  $M'$  has too, and the cardinality of its set of edges is  $O(\text{Card}(E))$ . For each  $k \in \mathbf{N}$ , such a real-valued machine  $M'$ , if it uses  $p$  processors, induces a  $k$ -th entropic

co-tree, which happens to be in a  $p$ -fold product of the AMC of algebraic computational trees and so can be rewritten (by flattening the said products) in a co-tree in the AMC of algebraic computational trees, but  $p$  times longer. By Theorem 32, this graphing defines a system of equations separating the integral inputs accepted by  $M$  from the others of total degree  $2kp \times 2^{O(\text{Card}(E))} \cdot 2^{h_0(|M|)}$ .

**Surfaces and fans.** An algebraic surface in  $\mathbf{R}^3$  is a surface defined by an equation of the form  $p(x, y, z) = 0$  where  $p$  is a polynomial. If  $S$  is a set of surfaces  $S_i$ , each defined by a polynomial  $p_i$ , the *total degree* of  $S$  is defined as the sum of the degrees of polynomials  $p_i$ .

Let  $K$  be a compact of  $\mathbf{R}^3$  delimited by algebraic surfaces and  $S$  be a finite set of algebraic surfaces of total degree  $\delta$ . We can assume that  $K$  is delimited by two affine planes of equation  $z = \mu$  and  $z = 2\mu_z$  and the cone of a rectangle  $\{(x, y, 1) \mid |x|, |y| \leq \mu_{x,y}\}$ , by taking any such compact containing  $K$  and adding the surfaces bounding  $K$  to  $S$ .  $S$  defines a partition of  $K$  by considering maximal compact subspaces of  $K$  whose boundaries are included in surfaces of  $S$ . Such elements are called the *cells* of the decomposition associated to  $S$ .

**Definition 39.** Let  $K$  be a compact of  $\mathbf{R}^3$ . A finite set of surfaces  $S$  on  $K$  separates a  $\rho$ -fan  $\text{Fan}$  on  $K$  if the partition on  $\mathbf{Z}^3 \cap K$  induced by  $S$  is finer than the one induced by  $\text{Fan}$ .

A major technical achievement of Mulmuley [Mul99] – not explicitly stated – was to prove the following theorem, of purely geometric nature.

**Theorem 40** (Mulmuley). *Let  $S$  be a finite set of algebraic surfaces of total degree  $\delta$ . There exists a polynomial  $P$  such that, for all  $\rho > P(\delta)$ ,  $S$  does not separate  $\rho$ -fans.*

## 6 The result

We can then obtain the following result, from which Theorem 1 is a straightforward corollary, by putting together Theorems 32, 37, and 40.

**Theorem 41.** *Let  $G$  be a deterministic graphing interpreting a PRAM with  $2^{O((\log N)^c)}$  processors, where  $N$  is the length of the inputs and  $c$  any positive integer.*

*Then  $G$  does not decide  $\text{maxflow}$  in  $O((\log N)^c)$  steps.*

The definition of PRAM we considered is quite idealized in that the complexities are stated without references to the size of the inputs. We can consider the *length* of an input to be the minimal length of a binary word representing it. To account for the size of the input, we can refine the model and consider that the memory locations do not contain natural numbers but binary words whose lengths can be a parameter of the model. The approach outlined here is very sensible to the fact that all computed quantities are polynomial in the inputs, which would be wrong if we allowed for access of any arbitrary bit of the memory. Mulmuley defines this model as the PRAM *model without bit operations*, by taking the point of view that the integers stored in the memory really are binary words, whose individual bits can not be accessed easily.

Here, we are able to compute arbitrary quotients by adding new phantom variables. Our model is thus of PRAM without bit operations and arbitrary



divisions, in Mulmuley’s parlance. The question of how this class is related to NC is open: indeed, while bit extractions cannot be performed in constant time by our machines (a consequence of Theorem 32), they can be simulated in logarithmic time.

## References

- [AKM65] R. L. Adler, A. G. Konheim, and M. H. McAndrew. Topological entropy. *Transactions of the American Mathematical Society*, 114(2):309–319, 1965.
- [BO83] M. Ben-Or. Lower bounds for algebraic computation trees. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC ’83, pages 80–86, New York, NY, USA, 1983. ACM.
- [Car83] P. J. Carstensen. *The Complexity of Some Problems in Parametric Linear and Combinatorial Programming*. PhD thesis, Ann Arbor, MI, USA, 1983.
- [Cuc92] Felipe Cucker.  $\mathbf{P}_r \neq \mathbf{NC}_r$ . *Journal of Complexity*, 8(3):230 – 238, 1992.
- [GSS82] L. M. Goldschlager, R. A. Shaw, and J. Staples. The maximum flow problem is log space complete for p. *Theoretical Computer Science*, 21:105–111, 1982.
- [Hof75] J. E. Hofer. Topological entropy for noncompact spaces. *The Michigan Mathematical Journal*, 21(3):235–242, 1975.
- [Mil64] John Milnor. On the Betti numbers of real varieties. In *Proceedings of the American Mathematical Society*, page 275, 1964.
- [Mul99] K. Mulmuley. Lower bounds in a parallel model without bit operations. *SIAM J. Comput.*, 28(4):1460–1509, 1999.
- [Mur80] K. G Murty. Computational complexity of parametric linear programming. *Mathematical programming*, 19(1):213–219, 1980.
- [Nef94] C. Andrew Neff. Specified precision polynomial root isolation is in NC. *Journal of Computer and System Sciences*, 48(3):429 – 463, 1994.
- [PS18a] Luc Pellissier and Thomas Seiller. Prams over integers do not compute maxflow efficiently. Submitted, 2018.
- [PS18b] Luc Pellissier and Thomas Seiller. Semantics, entropy and complexity lower bounds. Technical Report, <https://www.seiller.org/documents/articles/EntropyLowerBounds.pdf>, 2018.
- [Sei16] T. Seiller. Interaction graphs: Full linear logic. In *IEEE/ACM Logic in Computer Science (LICS)*, 2016.

- [Sei17] T. Seiller. Interaction graphs: Graphings. *Annals of Pure and Applied Logic*, 168(2):278–320, 2017.
- [Sei18] Thomas Seiller. Interaction graphs: Nondeterministic automata. *ACM Transaction in Computational Logic*, 19(3), 2018.
- [Sei19] Thomas Seiller. Interaction Graphs: Exponentials. *Logical Methods in Computer Science*, Volume 15, Issue 3, August 2019.
- [SY82] J. M. Steele and A. Yao. Lower bounds for algebraic decision trees. *Journal of Algorithms*, 3:1–8, 1982.
- [Tho65] R. Thom. *Sur l'homologie des variétés algébriques réelles*, pages 255–265. Princeton University Press, 1965.

## A Omitted proofs

*Proof of 23.* We show that the set  $\text{Adm}_k(G)$  of admissible sequences of states of length  $k$  has the same cardinality as the smallest subcover of  $\mathcal{S} \vee [G]^{-1}(\mathcal{S}) \vee \dots \vee [G]^{-(k-1)}(\mathcal{S})$ . Hence  $H^k(G, \mathcal{S}) = \frac{1}{k} \log_2(\text{Card}(\text{Adm}_k(G)))$ , which implies the result.

The proof is done by induction. As a base case, we consider the set of  $\text{Adm}_2(G)$  of length 2 admissible sequences of states and the cover  $\mathcal{V} = \mathcal{S} \vee [G]^{-1}(\mathcal{S})$  of  $D = [G]^{-1}(\mathbf{X})$ . An element of  $\mathcal{V}$  is an intersection  $\mathbf{X} \times \{s_1\} \cap [G]^{-1}(\mathbf{X} \times \{s_2\})$ , and is therefore equal to  $C[s_1, s_2] \times \{s_1\}$  where  $C[s_1, s_2] \subset \mathbf{X}$  is the set  $\{x \in \mathbf{X} \mid [G](x, s_1) \in \mathbf{X} \times \{s_2\}\}$ . This set is empty if and only if the sequence  $s_1 s_2$  belongs to  $\text{Adm}_2(G)$ . Moreover, given another sequence of states  $s'_1 s'_2$ , the sets  $C[s_1, s_2]$  and  $C[s'_1, s'_2]$  are disjoint. Hence a set  $C[s_1, s_2]$  is *removable from the cover*  $\mathcal{V}$  if and only if  $s_1 s_2$  is not admissible. This proves the case  $k = 2$ .

The step for the induction is similar. One considers the partition  $\mathcal{S}_k = \bigvee_{i=0}^{-(k-1)} [G]^i(\mathcal{S})$  as  $\mathcal{S}_{k-1} \vee [G]^{-(k-1)}(\mathcal{S})$ . By the same argument, one shows elements of  $\mathcal{S}_{k-1} \vee [G]^{-(k-1)}(\mathcal{S})$  are of the form  $C[\mathbf{s} = (s_0 s_1 \dots s_{k-1}), s_k] \times \{s_1\}$  where  $C[\mathbf{s}, s_k]$  is the set  $\{x \in \mathbf{X} \mid \forall i = 2, \dots, k, [G]^{i-1}(x, s_1) \in \mathbf{X} \times \{s_i\}\}$ . Again, these sets  $C[\mathbf{s}, s_k]$  are pairwise disjoint and empty if and only if the sequence  $s_0 s_1 \dots s_{k-1}, s_k$  is not admissible.  $\square$

*Proof of Lemma 24.* Let us fix some  $\epsilon > 0$ . Notice that if we let  $H_k(G, \mathcal{U}) = H^0(\mathcal{U} \vee [G]^{-1}(\mathcal{U}) \vee \dots \vee [G]^{-(k-1)}(\mathcal{U}))$ , the sequence  $H_k(\mathcal{U})$  satisfies  $H_{k+1}(\mathcal{U}) \leq H_k(\mathcal{U}) + H_1(\mathcal{U})$ . By Fekete's lemma on subadditive sequences, this implies that  $\lim_{k \rightarrow \infty} H_k/k$  exists and is equal to  $\inf_k H_k/k$ . Thus  $h([G], \mathcal{U}) = \inf_k H_k/k$ .

Now, the entropy  $h([G])$  is defined as  $\sup_{\mathcal{U}} \lim_{k \rightarrow \infty} H_k(\mathcal{U})/k$ . This then rewrites as  $\sup_{\mathcal{U}} \inf_k H_k(\mathcal{U})/k$ . We can conclude that  $h([G]) \geq \inf_k H_k(\mathcal{U})/k$  for all finite open cover  $\mathcal{U}$ .

Since  $\inf_k H_k(\mathcal{U})/k$  is the limit of the sequence  $H_k/k$ , there exists an integer  $N$  such that for all  $k \geq N$  the following inequality holds:  $|H_k(\mathcal{U})/k - \inf_k H_k(\mathcal{U})/k| < \epsilon$ , which rewrites as  $H_k(\mathcal{U})/k - \inf_k H_k(\mathcal{U})/k < \epsilon$ . From this we deduce  $H_k(\mathcal{U})/k < h([G]) + \epsilon$ , hence  $H^k(G, \mathcal{U}) < h([G]) + \epsilon$  since  $H^k(G, \mathcal{U}) = H_k(G, \mathcal{U})$ .  $\square$

*Proof of Proposition 31.* For a fixed sequence  $\vec{e}$ , the number of elements  $n_{\vec{e}}^{\pi}$  of length  $m$  in  $\text{COT}_k(T)$  is bounded by the number of elements in the  $m$ -th cell decomposition of  $T$ , and is therefore bounded by  $g(m) = 2^{m \cdot h_0([T])}$  by Theorem 28. The number of sequences  $\vec{e}$  is bounded by  $\text{Card}(\text{Seq}_k(E))$  and therefore the size of  $\text{COT}_k(T)$  is thus bounded by  $\text{Card}(\text{Seq}_k(E)) \cdot 2^{(k+1) \cdot h_0([T])}$ .  $\square$

*Proof of Corollary 35.* Let  $T$  be an algebraic computation tree computing the membership problem for  $W$ , and consider the computational treeing  $[T]$ . Let  $d$  be the height of  $T$ ; by definition of  $[T]$  the membership problem for  $W$  is computed in exactly  $d$  steps. Thus, by the previous theorem,  $W$  has at most  $\text{Card}(\text{Seq}_k(E)) \cdot 2^{h_0([T]) + 1} 3^{2d+n+1}$  connected components. As the interpretation of an algebraic computational tree,  $h_0([T])$  is at most equal to 2, and  $\text{Card}(\text{Seq}_k(E))$  is bounded by  $2^d$ . Hence  $N \leq 2^d \cdot 2^3 3^{n+1} 3^{2d}$ , i.e.  $d = \Omega(\log N)$ .  $\square$

*Proof of Lemma ??.* Let

$$W = \left\{ x \in \mathbf{R} \mid \left\lfloor \frac{x}{2^{k-1}} \right\rfloor - 2 \left\lfloor \frac{x}{2^k} \right\rfloor = 1 \right\}$$

$W$  is the disjoint union of  $m - k + 1$  intervals, and so is its complement in  $]0; 2^m[$ . So, by Theorem 35, any algebraic computation tree computing the  $k$ -th bit has height  $\Omega(\log m - k)$ .  $\square$

*Proof of Theorem 32.* If  $G$  computes the membership problem for  $W$  in  $k$  steps, it means  $W$  can be described as the union of the subspaces corresponding to the nodes  $n_{\vec{e}}^{\pi}$  with  $\pi$  of length  $k$  in  $\text{COT}_k(T)$ . Now, each such subspace is an algebraic set, as it can be described by a set of polynomials as follows.

We define a system of equations  $(E_i^{\mathbf{e}})_i$  for each node  $n_{\vec{e}}^{\pi}$  of the entropic co-tree  $\text{COT}_k(T)$ . This is done inductively on the size of the path  $\vec{e}$ , keeping track of the last modifications of each register. I.e. we define both the system of equations  $(E_i^{\mathbf{e}})_i$  and a function  $\mathfrak{h}(\mathbf{e}) : \mathbf{R}^{\omega} + \perp \rightarrow \omega$  (which is almost everywhere null)<sup>7</sup>. For an empty sequence, the system of equations is empty, and the function  $\mathfrak{h}(\mathbf{e})$  is constant, equal to 0.

Suppose now that  $\vec{e}' = (e_1, \dots, e_m, e_m + 1)$ , with  $\vec{e} = (e_1, \dots, e_m)$ , and that one already computed  $(E_i^{\mathbf{e}})_{i \geq m}$  and the function  $\mathfrak{h}(\mathbf{e})$ . We now consider the edge  $e_{m+1}$  and let  $(r, r')$  be its realizer. We extend the system of equations  $(E_i^{\mathbf{e}})_{i \geq m}$  by a new equation  $E_{m+1}$  and define the function  $\mathfrak{h}(\mathbf{e}')$  as follows:

- if  $r = \text{add}_i(j, k)$ ,  $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x) + 1$  if  $x = i$ , and  $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x)$  otherwise; then  $E_{m+1}$  is  $x_i^{\mathfrak{h}(\mathbf{e}')(i)} = x_j^{\mathfrak{h}(\mathbf{e}')(j)} + x_k^{\mathfrak{h}(\mathbf{e}')(k)}$ ;
- if  $r = \text{sub}_i(j, k)$ ,  $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x) + 1$  if  $x = i$ , and  $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x)$  otherwise; then  $E_{m+1}$  is  $x_i^{\mathfrak{h}(\mathbf{e}')(i)} = x_j^{\mathfrak{h}(\mathbf{e}')(j)} - x_k^{\mathfrak{h}(\mathbf{e}')(k)}$ ;
- if  $r = \text{mult}_i(j, k)$ ,  $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x) + 1$  if  $x = i$ , and  $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x)$  otherwise; then  $E_{m+1}$  is  $x_i^{\mathfrak{h}(\mathbf{e}')(i)} = x_j^{\mathfrak{h}(\mathbf{e}')(j)} \times x_k^{\mathfrak{h}(\mathbf{e}')(k)}$ ;
- if  $r = \text{div}_i(j, k)$ ,  $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x) + 1$  if  $x = i$ , and  $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x)$  otherwise; then  $E_{m+1}$  is  $x_i^{\mathfrak{h}(\mathbf{e}')(i)} = x_j^{\mathfrak{h}(\mathbf{e}')(j)} / x_k^{\mathfrak{h}(\mathbf{e}')(k)}$ ;

<sup>7</sup>The use of  $\perp$  is to allow for the creation of fresh variables not related to a register.

- if  $r = \text{add}_i(c, k)$ ,  $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x) + 1$  if  $x = i$ , and  $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x)$  otherwise; then  $E_{m+1}$  is  $x_i^{\mathfrak{h}(\mathbf{e}')(i)} = c + x_k^{\mathfrak{h}(\mathbf{e}')(k)}$ ;
- if  $r = \text{sub}_i(c, k)$ ,  $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x) + 1$  if  $x = i$ , and  $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x)$  otherwise; then  $E_{m+1}$  is  $x_i^{\mathfrak{h}(\mathbf{e}')(i)} = c - x_k^{\mathfrak{h}(\mathbf{e}')(k)}$ ;
- if  $r = \text{mult}_i(c, k)$ ,  $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x) + 1$  if  $x = i$ , and  $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x)$  otherwise; then  $E_{m+1}$  is  $x_i^{\mathfrak{h}(\mathbf{e}')(i)} = c \times x_k^{\mathfrak{h}(\mathbf{e}')(k)}$ ;
- if  $r = \text{div}_i(c, k)$ ,  $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x) + 1$  if  $x = i$ , and  $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x)$  otherwise; then  $E_{m+1}$  is  $x_i^{\mathfrak{h}(\mathbf{e}')(i)} = c/x_k^{\mathfrak{h}(\mathbf{e}')(k)}$ ;
- if  $r = \text{sqrt}_i(k)$ ,  $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x) + 1$  if  $x = i$ , and  $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x)$  otherwise; then  $E_{m+1}$  is  $x_i^{\mathfrak{h}(\mathbf{e}')(i)} = \sqrt{x_k^{\mathfrak{h}(\mathbf{e}')(k)}}$ ;
- if  $r = \text{Id}$ , the source of the edge  $e_q$  is of the form  $\{(x_1, \dots, x_{n+\ell}) \in \mathbf{R}^{n+\ell} \mid P(x_k)\} \times \{i\}$ .
  - If  $P(x_k)$  is  $x_k \neq 0$ ,  $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x) + 1$  if  $x = \perp$ , and  $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x)$  otherwise then  $E_{m+1}$  is  $x_{\perp}^{\mathfrak{h}(\mathbf{e}')(i)} x_k^{\mathfrak{h}(\mathbf{e}')(k)} - 1 = 0$ ;
  - otherwise we set  $\mathfrak{h}(\mathbf{e}') = \mathfrak{h}(\mathbf{e})$  and  $E_{m+1}$  equal to  $P$ .

We now consider the system of equations  $(E_i)_{i=1}^k$  defined from the path  $\mathbf{e}$  of length  $k$  corresponding to a node  $n_{\mathbf{e}}^{\pi}$  of the  $k$ -th entropic co-tree of  $G$ . This system consists in  $k$  equations of degree at most 2 and containing at most  $k + n$  variables, counting the variables  $x_1^0, \dots, x_n^0$  corresponding to the initial registers, and adding at most  $k$  additional variables since an edge of  $\vec{e}$  introduces at most one fresh variable. Since the number of vertices  $n_{\mathbf{e}}^{\pi}$  is bounded by  $\text{Card}(\text{Seq}_k(E)) \cdot 2^{k \cdot h_0([G])}$  by Theorem 31, we obtained the stated result.  $\square$

*Proof of Proposition 38.* To compute  $p//q$ , where  $p, q \in \mathbf{Z}$ , consider the real-valued machine such that the  $i^{\text{th}}$  processor computes  $x = p/q - i$  and if  $0 < x \leq 1$ , writes  $i$  in the shared memory. This operation generalizes euclidian division and is computed in constant time. Moreover, this only uses a number of processor linear in the bitsize of the inputs if they are integers.  $\square$

*Proof of Theorem 41.* Let  $n = N^{2^c}$  and consider the problem 37 of Thm. 37 and its family of affine parametrizations of bitsize  $O(n^2) = O(N^{4^c})$  and complexity  $\rho(n) = 2^{\Omega(n)} = 2^{\Omega(N^{2^c})}$ .

By Section 5.2, we know that the partition induced by  $G$  after  $O((\log N)^c)$  can be defined by a set of equations of total degree  $2O((\log N)^c)2^{O((\log N)^c)} = 2^{O((\log N)^c)}$ .

Let  $P$  be the polynomial of Thm. ???. For large enough values of  $N$ ,  $\rho(n)$  is larger than  $P(2^{O((\log N)^c)}) = 2^{O((\log N)^c)}$ . So,  $G$  does not decide maxflow in  $O((\log N)^c)$  steps.  $\square$